Report No. 7177

Distributed Heuristics for Maintaining Connectivity in Mobile Networks (SRNTN-54)

Robert Willis

Prepared By:

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138

Prepared for:

DARPA/ISTO
1400 Wilson Bl.
Arlington, VA. 22209

Sponsored by:

The Defense Advanced Research Projects Agency
Under Contract No. MDA-903-83-C-0131

Contents

Abstract

In a packet radio network, a packet radio (PR) can establish links to any other PR within hearing range, but, if too many PRs are nearby, its link table may overflow. To maintain connectivity and keep their link-table sizes feasible it is necessary for PRs to choose a subset of all possible links. This selection problem resembles the problem of finding spanning trees in a distributed manner. We develop a distributed heuristic for choosing which links should be retained to keep the network connected. Since the heuristic uses only local information, it cannot guarantee to keep the network connected in those cases that require more global information. The heuristic is of interest because it requires less coordination, less information and, therefore, lower communications overhead than algorithms that guarantee to maintain connectedness.

1. Introduction

In this paper, we explore some of the issues involved in choosing which connections to maintain in a dense packet-radio (PR) network. Specifically, the goal is to find algorithms or heuristics that maintain a richly connected network whenever possible or practical. This paper presents a collection of heuristics from which some subset may be chosen for implementation in the SURAN PR network[4].

In a PR network, the density of PRs in some local geographic region may be higher than is desirable. This density may or may not be temporary. Dense environments cause problems because too many PRs can communicate directly among themselves. If the density is high enough, a PR may even run out of internal table space for maintaining local connections – particularly if it has a limited table size or a fixed amount of random-access memory.

To conform to table space limitations, the PRs must choose which connections to maintain. If there were no constraints other than computational feasibility, the choices would usually be easy to make. The PRs could compute a spanning tree for the network to maintain connectedness, and then add in other links to obtain rich connectivity. However, the SURAN PRs are memory limited. Other constraints are limited control-data bandwidth, the requirement to operate with and without time slots, the requirement to have no single point of failure, and the requirement to keep the network connected in spite of the mobility of PRs. Taken together, these constraints transform [illegible]ly simple problem into a rather difficult one.

Some solutions to this problem are derived in th[illegible]aper. All solutions use only local connectivity information in an explicit manner. Section 2 contains definitions of generally useful terms; other terms are defined as required. Section 3 defines the problem. Section 4 derives heuristic solutions that work most of the time, and Section 5 speculates on a back-up procedure that may maintain connectedness even when the primary methods fail.

2. Terminology

A *link* is defined as a bidirectional one-hop path between two PRs. For a a link to exist between PRs A and B, PR A must be able to hear PR B, and PR B must be able to hear PR A. Throughout this paper, the term link also will imply that a reasonable amount of communication can take place between the two PRs – that is, the bandwidth is sufficient to implement the protocols

Two PRs are defined to be *adjacent* if, and only if, a link could or does exist between them.

Two PRs are *good neighbors* if they are adjacent, if each is aware of the existence of the other, and if each agrees that a link exists between them; otherwise, they are *bad neighbors*.

In a dense region, sometimes one PR, A, will decide to establish communication with an adjacent PR, Q, at the expense of some other PR, B, which is currently a good neighbor of A. In this case, before PR A drops PR B, PR Q will be called the *candidate* PR. At such time as PR A decides that the candidate PR should become a good neighbor at the expense of B, PR A is said to *exclude* PR B. PR B has been *excluded* and is the *excludee*. Furthermore, PR A is said to be *active* from the moment that it makes known that it is currently deciding on whom to exclude until the time that it notifies its neighbors about which PR has been excluded. When a PR is not active, it is *passive*.

Finally, throughout this paper, the variable L ($L > 2$) represents the maximum number of entries in a PR's local connection table. This table is referred to as the neighbor table. Note that L does not have to be identical for all PRs, although for simplicity we will assume it is so. This assumption has no effect on any of the algorithms or heuristics in this paper, although some of the observations would have to be slightly restated if L were not identical for all PRs.

3. Problem Definition

3.1 Maximum Degree Constraint

Consider the packet radio network as a graph G. Each packet radio (PR) corresponds to one node v in G. Each link in the packet radio network corresponds to an edge e. The undirected graph $G(V, E)$ represents our PR network with V as the set of all nodes (or PRs) and E as the set of edges (or links). We do not allow self-links at any node in G.

Let the function $degree(v)$, $v \in V$, be equal to the number of links that terminate (or, equivalently, begin) at node v. The $degree(v)$ for node v is simply the number of nodes adjacent to v. Let $M = max[degree(v)]$, over all $v \in V$. If $M \leq L$ then the packet radios have enough table space in the neighbor table for all connections, even in the densest neighborhoods. In this case all adjacent PRs are good neighbors.

Only in the case when $M > L$ must one or more PRs choose which connections should appear in their tables. In this case they must decide which subset of adjacent PRs will be good neighbors.

3.2 Algorithmic Issues

Any algorithm for selecting adjacent PRs for good neighbors should have the following property: If G is connected then G', the graph obtained by deleting various edges in G to satisfy the constraint that no node has a degree greater than L, should also be connected. In general, G' will not be unique. Many graphs may be obtained that satisfy the degree constraint. However, there might not be a connected subgraph of maximum degree L. In this case, no algorithm could succeed in connecting the revised graph.

Transforming G to a G' that satisfies the maximum degree constraint is not terribly difficult, provided that information about all the nodes and edges is available. A simple way to effect the transformation would be to find a spanning tree and add back as many other links as possible without violating the degree constraint. Of course, some spanning trees may violate the degree constraint themselves. The kind of tree that serves best is one with only a few edges per node.

Implementing such a transformation procedure in the SURAN network, though, has several practical difficulties.

First, the computation must be distributed, not centralized. One way to accomplish this is by distributing global information to all PRs and having each PR run the same centralized algorithm. However, a true distributed algorithm would be more desirable since it would reduce the amount of information that must be conveyed to each node. Truly distributed algorithms, though, have their own protocols which must be implemented in the PR. These protocols will use up more of the memory that we are trying to conserve.

Second, communication is unreliable in the PR network. Since distributed spanning tree algorithms[1,2] seem to require reliable, order-preserving, error-free communication, simple broadcasting is inappropriate. A reliable link mechanism must be used, but this will impede the timely transfer of information. In a static network, this delay is unimportant since this computation will not be performed very often. In a mobile network, though, a delay could prevent the algorithm from converging to an acceptable solution before the configuration changes again.

Third, in the PR network, not all of the possible links are available simultaneously, since the degree constraint limits each node to a maximum of L. This may complicate matters since some nodes may be left out of the spanning tree computation. Ideally, the algorithm would know about all the possible links in the network. In this case, it is likely that information about all of the possible links may need to be stored somewhere for some period of time. If it were stored in a distributed manner, the information would be kept at each node. Thus, unless it is possible to query PRs individually across the net, it seems that the minimum amount of state information that must be kept at each node would be the purely local information for that node: the one-hop links in the G network. Keeping this information, though, means that there is space for listing all the links of a neighbor – space for M entries as opposed to L.

We believe that, given limited memory, it is impractical to implement an algorithm that transforms G to a connected G' subject to the degree constraint and guarantees to do so in a fixed, but arbitrary, amount of time independent of the size of G. It should be apparent that implementation of such an algorithm will use up a sizable amount of memory for the distributed spanning tree (or some similar algorithm) and its protocols. In addition, user bandwidth is reduced, since the edge information must be sent reliably around the network. If the largest foreseeable M requires less memory than the storage requirement implementing the above protocols, it would be better to increase L to M than to implement the protocols.[1]

What happens if M is much larger than L, and there is a limited amount of memory available? If we are willing to give up the *guarantee* of maintaining connectedness, there are other alternatives that tend to keep the network connected and don't require global network information. These alternatives are discussed in the next section.

[1] If, however, the network edge information is already available or is needed for some other reasons (e.g., routing or flow control), it may make sense to implement this algorithm. This may be advantageous later when you realize that the "largest foreseeable M" was too small.

4. Local Heuristics

4.1 Overview and Assumptions

4.1.1 Overview

In this section we propose heuristics for maintaining connected networks. The heuristics are all based upon the assumption that only local information is available. The local information is obtained from neighbors to which a node has a current connection. Local information may also consist of information based on computations performed by a node's neighbors. We assume that it is feasible to store a small quantity of information calculated by neighbor nodes, but it is not feasible to store each neighbors' list of neighbors. Each node periodically broadcasts a list of all of its good neighbors, and its candidate (defined in Section 2), if any. The packet in which this (and other) information is made available is called a PROP (Packet Radio Organization Packet), in accordance with current SURAN packet radio terminology.

Section 4.2 contains some lemmas about graph connectivity that are useful for designing heuristics. Although the lemmas become increasingly more complex and therefore generate heuristics that require more coding to implement, the increasing complexity does increase the number of network configurations that will remain connected. Section 4.3 addresses stability and convergence issues. Section 4.4 presents some ways one might speed up convergence. Section 4.5 explains how to implement some of the details in the SURAN PR network.

In Section 4.2, we assume that all packets are received by intended recipients on the first transmission attempt. In Section 4.5, this restriction is removed. The material in Section 4.2 also contains the assumption that the execution of the heuristic is perfectly synchronized at all nodes. This synchronization takes a form that is explained in the next paragraph. This restriction is also removed in Section 4.5.

Whenever the heuristic needs to be run, a major cycle is begun. All nodes begin the major cycle simultaneously. Once the major cycle has begun, no node will add any new nodes to its neighbor table, unless the new node was a candidate at the beginning of the major cycle. The completion of the major cycle occurs whenever all nodes are passive. At this point, the network may accept new links until the start of another heuristic cycle. A minor cycle is a period in which neighbors exchange information among themselves. This is basically equivalent to a PROP cycle. This nomenclature is not used until Section 4.3.

Many of the issues that arise in this context of heuristics are subtle and often elusive. The utilization of set

notation and graph nomenclature has been necessary to avoid either understatements or overgeneralizations when stating results. We have rephrased some of the results in less precise but more accessible terms in those instances where the mathematical formulation is hard to follow.

4.1.2 Assumptions

Each node has a neighbor table that can hold L entries plus one overflow entry. Nodes which have neighbor tables with fewer than L entries in use will accept any link offered to them. If a node has a table with L entries in use, it must be more discriminating when accepting an additional link because it will have to delete a link that is already in the table. When it is necessary to replace a link in the neighbor table, the new link should be ready to function as soon as it is entered into the table. Therefore one additional entry is provided, in order to obtain a smooth transition. The neighbor that fills this special entry is called the candidate. The link with the candidate is not a good (bi-directional) link; neither node will consider the other as a good neighbor. If this link exists, the node is said to have $L + 1$ links, even though one is only a potential link.

Since the deletion of a link can cause routing changes to propagate throughout the network, the deletion of a link in the table should only be done when absolutely necessary. When a node, Q, has L entries in use in its neighbor table and hears from a node, R, to which it currently does not have a link, Q will determine whether to consider R as a candidate. If Q does not already have a candidate, the following rule is used.

Candidate Determination Rule:

> Q will consider R as a candidate if, and only if, Q cannot route to R.

Q is said to be able to route to R if R appears in the routing table and the route is marked good.[1] Therefore, if Q's neighbor table is full, it will not even consider establishing a link to a node that it can reach via some other path.

A node can have only one candidate. If Q has a candidate, PROPs received from any node that is neither one of Q's good neighbors nor Q's candidate are ignored by Q.

Once a node has a candidate, it has a decision to make: either it must remove an entry for some other node from the neighbor table and substitute the candidate in its place, or it must decide to remove the candidate from the candidate slot. The following modus operandi is critical. When one node, Q, actually excludes another node S, then Q expunges (deletes) S from its neighbor table, freeing up one entry. The candidate R is stored immediately in the just-freed entry. *When S receives a PROP from Q indicating that Q has excluded S, S expunges Q from S's neighbor table, and also frees up one entry.* Note that if S originally had $L + 1$ entries (including the candidate) and had not (yet) been able to make a decision

[1] Implementation Note: If R does not appear in the routing tables, it is recommended that Q also look in R's PROP to see whether any of Q's current good neighbors appear there.

as to which of its neighbors to exclude, then S will not have to make *any* decision when it hears that Q has excluded it. S will have L links, since it will remove the Q entry and substitute its own candidate for that entry.

4.2 Decision Rules

For the reasons stated earlier, connectedness cannot be guaranteed in SURAN, since the cost in memory and bandwidth is very high. However, it is possible to develop heuristics which will tend to keep the network connected in most cases. In order to keep the amount of information that must be sent around the network small, the information that the heuristics may use is restricted to information obtainable from the routing algorithm and local information. Local information is defined to be information found in a neighbor's PROP.

The guidelines that follow should provide a reasonable strategy for maintaining connectedness in a network. This overview provides a framework in which to place the heuristics that are developed subsequently. Exact definition of terms used in the guidelines and suggestions on how to implement the procedures in the PR come later. For now, though, assume that an expungable neighbor is one that appears in the neighbor table of a PR and is not currently of use to that PR.

Guidelines

Immediately upon receipt of a PROP from a PR that does not appear in the neighbor table of PR X, PR X must take one of the following actions:

1. If X's neighbor table is not full, add the PR to X's neighbor table.
2. Or, if there is one or more expungable neighbors, select one and expunge (delete) it. The new PR takes that table slot.
3. Or, if there are no expungable neighbors and if X currently has a good route to the new PR, ignore the PR. Use the routing tables to determine if a good route exists.
4. Or, if there is not a good route to the new PR, accept this PR as a candidate. Allow the link qualities a reasonable amount of time to build to acceptable values before you actually exclude someone. At that point, if there are no empty slots and no expungable neighbors, find a neighbor to exclude by using the primary and extended selection rules defined later in this section.

Let us return to our formulation of the problem in terms of the graph G, and its (generally non-unique) solution G'. We seek rules that allow decisions to be made based primarily upon information local to a node.

To begin, we cite a concept specified by Bill Miller[3]. Miller found that correct decisions about which link to discard could easily be made on the basis of localized information in certain cases. An inexact and

incomplete rendition of his rule is "if you have to exclude someone, exclude a good neighbor of yours which has the most (good) neighbors in common with you." Unfortunately, the specification given in the referenced paper (even as later modified in an addendum) manages to produce undesirable selections in some rather simple graphs. For example, given the configuration in Figure 4.1, his rules can lead to the state where A will permanently exclude B, and C permanently excludes A. It is certainly true that his rules can be modified to fix this case, but taking a slightly more general approach will solve some other problems as well.

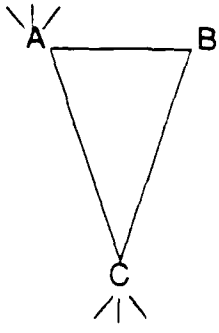

Figure 4.1: Assume $L = 4$, and $\{B,C\}$ or $\{A,B\}$ are best candidates for exclusion for A and C respectively.

The basic problem that can occur in Figure 4.1 is that PRs A and C both need to make a decision to bring the number of links down to four, the limit for this example. Unfortunately, the decision that each PR makes invalidates some assumptions held by the other. In this case, the invalidated assumption is that A, B, and C are neighbors.

In what cases can we make decisions without invalidating assumptions held by others? Some reflection leads to Lemma 4.1, which says in graphic terms that PR Q can safely remove a link with some PR that shares a neighbor with Q without causing the network to become disconnected if each of Q's neighbors is passive.

Lemma 4.1 *Assume a PR (say Q) needs to remove a link to satisfy the maximum degree constraint. Let $\mathcal{N}$ be the set of nodes consisting of Q's good neighbors, and let $m = max[degree(v)]$, for all $v \in \mathcal{N}$. If $m \leq L$, then PR Q can unilaterally remove a link without causing any neighbor to become disconnected, assuming that there is at least one $v \in \mathcal{N}$ that shares a neighbor with PR Q.*

Note: When a PR obtains the $L+1^{st}$ link, it must decide to exclude one link to keep the limit within L. That is why $m \leq L$, and not $m < L$.

Further reflection discloses that this lemma is overly restrictive; the conditions set forth are sufficient, but not necessary. More general lemmas can be obtained.

Lemma 4.2, set forth next, is difficult to understand because it is stated in set-theoretic terms. However, the sets it defines are used extensively throughout the remainder of this paper. As an aid to understanding, it is first stated here in terms of active and passive nodes, as defined in Section 2. The following definitions are synonymous with the ones stated earlier. An active PR is one whose current degree is equal to $L+1$. A passive PR is one whose current degree is less than or equal to L. Lemma 4.2 simply states that PR Q can safely exclude a neighbor S if both PR S and its neighbors held in common with PR Q are all passive.

Lemma 4.2 *Assume a PR (say Q) needs to remove a link to satisfy the maximum degree constraint. Suppose that its initial choice is (good neighbor) PR S. Let $\mathcal{N}$ be the set of Q's good neighbors. Let the $\mathcal{NIC}$ (Neighbors In Common) set for Q and S be the set of nodes where node $v \in \mathcal{NIC}$ if and only if: $v \in \mathcal{N}$, and v and S are good neighbors. Thus, the set $\mathcal{NIC}$ for S – also written $\mathcal{NIC}(Q,S)$ – is the set of nodes that are good neighbors to both Q and S. Let $\mathcal{N}_1 = \{S\} \cup \mathcal{NIC}(Q,S)$. Finally, let $mn_1 = max[degree(v)]$, for all $v \in \mathcal{N}_1$. If $mn_1 \leq L$, then PR Q can unilaterally remove the link to S without causing any neighbor to become disconnected if and only if the set $\mathcal{NIC}$ is not empty.*

It is clear that this lemma is not as restrictive as the previous one when $\mathcal{N}_1$ is a proper subset of $\mathcal{N}$. Generally, $\mathcal{N}_1$ will be a proper subset of $\mathcal{N}$.

Lemma 4.2 broadens the number of cases in which a unilateral decision is safe to make. Unfortunately, it does not apply in the example of Figure 4.1 mentioned earlier, because mn_1 equals $L+1$ (i.e., there is an active node other than the node making the decision). Let us then consider lemma 4.3.

Lemma 4.3 *All of the definitions in lemma 4.2 are included here by reference. The set $\mathcal{NIC}$ is assumed to be non-empty. If PR S is a passive node, and at least one node in the set $\mathcal{NIC}(Q,S)$ is also passive, then it is safe for PR Q to unilaterally discard the link between Q and S.*

Proof: If PR Q excludes PR S, a path still exists between Q and S by going through the passive neighbor(s) in the set $\mathcal{NIC}$. Thus Q and the set of nodes in $\mathcal{N}_1$ are still connected, regardless of any single exclusion made by each of the active nodes in the set $\mathcal{NIC}$.

Lemma 4.3 broadens even more the number of cases in which a unilateral decision is allowed since it allows some of the nodes in the set $\mathcal{NIC}$ to be active. However, it still does not apply to that troublesome example since there is only one passive node among A, B, and C, and the lemma requires two.

Therefore, we state lemma 4.4, which does apply to the example.

Lemma 4.4 *All of the definitions in lemma 4.2 are included here by reference. The set $\mathcal{NIC}$ is assumed to be non-empty. If PR S is a passive node, and all nodes in the set $\mathcal{NIC}$ are active, then it is impossible to unilaterally remove the link between Q and S and guarantee that the subgraph consisting of Q and the set of nodes in $\mathcal{N}_1$ will remain connected knowing that the other nodes in the set $\mathcal{NIC}$ must make a choice as well.*

Proof: If all of the nodes in the set NIC exclude PR Q, or if all of them exclude PR S, then the subgraph is not connected.

Although lemma 4.4 applies to the example, it does not explicitly help. However, the solution is close at hand. Consider three nodes which are totally interconnected (see Figure 4.1). Further assume that two of these nodes are active (A and C), and the other (B) is passive. A, B, and C may have connections to an arbitrary number of other nodes. Suppose that C *must* exclude either A (active) or B (passive). C knows that B will do nothing – it is a passive node. Suppose C also knows that A must exclude some PR, but C has no idea which one A will exclude. What should C do?

From the point of view of C, A can make one of three decisions:

1. A can exclude some other node entirely. In this case, it doesn't matter whether C excludes A or B. The three nodes, A, B, and C, will remain connected.
2. A can exclude C. In this case, A should make the same kind of decision that C makes since the viewpoints are equivalent: An active node [now A] must choose to exclude either a passive node [B] or another active node [C]. Clearly the decision made should be one which would leave the nodes connected. The choice is obvious. Exclude the active node. This is a satisfyingly symmetric action, since C will (unilaterally) exclude A, while A will (unilaterally) exclude C.
3. A can exclude B. In view of the situation obtained from the second type of decision, it makes sense to never allow A (or C) to make this kind of decision.

Thus we arrive at lemma 4.5.

Lemma 4.5 *All of the definitions in lemma 4.2 are included here by reference. Assume that the set $NIC(Q,S)$ is non-empty, PR S is a passive node, and all nodes in the set NIC are active. If Q excludes any node S' from the set NIC instead of node S, then Q and the set of nodes N_1 will remain connected.*

Proof: Stated above.

Corollary 4.5 *Corollary:*

If PR S is an active node, and at least one node in the set NIC is passive, then exclude S. PR Q and the set of nodes N_1 will remain connected.

Lemma 4.5, together with its corollary explicitly solve the problem mentioned earlier.

One case remains that has not been covered by the current set of lemmas. What happens when a node has several neighbors in common with you, but they are all active? Further reflection leads to lemma 4.6, which will resolve most problems that arise when all nodes in the set N_1 are active.

Lemma 4.6 *The set nomenclature used here is compatible with that of lemma 4.2. If all nodes in some set N_1 are active and there is a neighbor S in N_1 which has two or more good neighbors in common with Q, [see Figure 4.3] then it is always possible to remove links in such a way as to keep the graph connected. Equivalently, if the cardinality of the set N_1 is three or greater, then it is possible to keep the graph connected.*

Proof: Keep the link between the PR S and Q, and ensure that S and Q exclude different neighbors in the set $NIC(Q,S)$. With this proviso, the decisions made by the remaining neighbors in the set NIC are irrelevant. [In Figure 4.3, Q would exclude either B or E, and S would exclude the other one.]

Note that this lemma does not apply to the case where the cardinality of the set N_1 is two or less. No unilateral or negotiated decision can help in this case.

We are now in a position to state the Primary and Extended Selection Rules mentioned in the Guidelines at the beginning of this section.

The Primary Selection Rule is consulted first when it is necessary to select a node for exclusion. Unlike Miller's selection rule [op cit.], the rule we use does not select the neighbor with the highest number of good neighbors in common. Why? Suppose that we used Miller's rule. Consider the case where the selected neighbor is an active node, and none of the nodes in the set NIC are passive. In that case all nodes in set N_1 are active. Certainly, we would be much better off finding an N_1 set that contains at least one passive node, even if it happens that the PR that we choose to exclude in that case has fewer neighbors in common with Q than the original choice.

Primary Selection Rule:

> Let a PR, Q, choose the PR S with the highest number of passive neighbors in common with Q as the PR for exclusion.

Note that our new selection rule incorporates lemmas 4.2, 4.3, and 4.5 by definition. Now, if it is the case that none of Q's neighbors has any passive neighbors in common with Q, we use the extended selection rule.

Extended Selection Rule:

> One of the following must be true; perform the associated action:
>
> 1. None of Q's neighbors has any neighbors in common with Q. As far as Q can determine, any decision that Q makes will partition the network. The state of the neighbor nodes is irrelevant in this case. [See Figure 4.2]
>
>   Action: Exclude the candidate in order to maintain routing stability.
> 2. At least two of Q's neighbors have at least one good neighbor in common with Q. All of them will be active.
>
>   In this case, one of the following must be true:

(a) Q has at least one neighbor with at least two good neighbors in common with Q. [See Figure 4.3]
Action: Use lemmas 4.6 and 4.7 to resolve the problem. Lemma 4.7 is explained subsequently. In Section 4.4, the 2-cycle resolution process, which is based upon these two lemmas, is shown to be a superior method for resolution.

(b) Q has two or more neighbors with one neighbor in common with Q, but no neighbors with more than one neighbor in common with Q. If none of these active neighbors goes passive, then any action Q takes could partition the network. [See Figure 4.4]
Action: Do not exclude anyone; wait indefinitely. "Wait" means that the PR repeatedly executes guidelines (2), (3), and (4). If the 2-cycle resolution process is used, indicate the $R.V$ node as NULL. While counterintuitive, it is the correct action (explained in lemma 4.7). It is roughly equivalent to not making a decision (the candidate will be frozen out) until someone else makes a decision, or a state change occurs. Either some one of the three nodes (maybe yourself) will eventually go passive – leading to a solution – or none of the three nodes ever will.

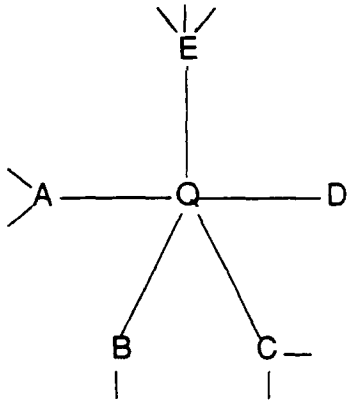

Figure 4.2: (L = 4. Q active; A, B, C, D, E states irrelevant. Candidate can be any one of A, B, C, D, E.)

There is one point that has not yet been touched upon. Lemma 4.6 depends upon the fact that nodes Q and S agree to exclude different neighbors in the set $NIC(Q, S)$. As yet, no mention has been made about how Q and S figure out that they need to make a joint decision. Further investigation reveals that this is not just an implementation detail; maintaining connectedness in the network depends upon proper application of lemma 4.6. Also, the discussion on convergence in the next section will rely on some concepts that are implicit in lemma 4.6. Hence, the following discussion and lemma.

Lemma 4.6 lends itself to easy implementation in a distributed environment. Given that S and Q mutually recognize each other (lemma 4.7), they can easily ensure that they exclude different neighbors without

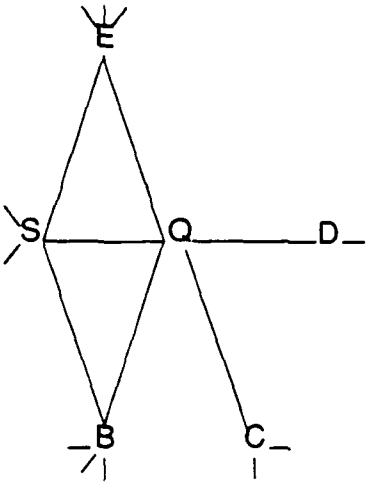

Figure 4.3: $L = 4$. Q, S, B, E active; C, D states irrelevant. Q has neighbor S that satisfies 2a

further communication. Both S and Q know that all nodes in the set NIC (which is now the same set for both PR S and Q) are currently active. Since S and Q have 2 or more good neighbors in common, they will use lemma 4.6 to resolve the deadlock. S and Q will not exclude each other. Q and S each order the nodes in the set NIC from high to low based on some immutable metric (such as ID number). Q and S also order Q and S. Have the larger of nodes S and Q exclude the largest of the nodes in the set NIC. Have the smaller of the nodes S and Q exclude the second-largest of the nodes in the set NIC.[2]

After these exclusions take place, all remaining nodes in the set NIC will be able to make independent decisions on which nodes to exclude since S and Q will both appear passive. The two excluded nodes must wait to be excluded since they don't have enough information to be able to independently compute that they would be the nodes chosen by S and Q.[3]

Other nodes in the set NIC may also have to wait for S and Q by the same reasoning. However, this wait requirement for extended selection rule part 2a dovetails nicely with the action that must be taken in part 2b.

The final task is to determine how S and Q will recognize each other. This is an important determination because an active node may have several possibilities for S. It matters which S is chosen by Q because we need to guarantee that some S and Q in the network will recognize each other. Lemma 4.7 will show that one can arrange the selection so some pair will recognize each other.

[2]Implementation note: Because of unreliable communication, excluding the smallest instead of the second-largest node should be safer.

[3]N.B. NIC nodes could compute whether or not they would be excluded by S and Q if all nodes kept a list of all of their immediate neighbors' neighbors.

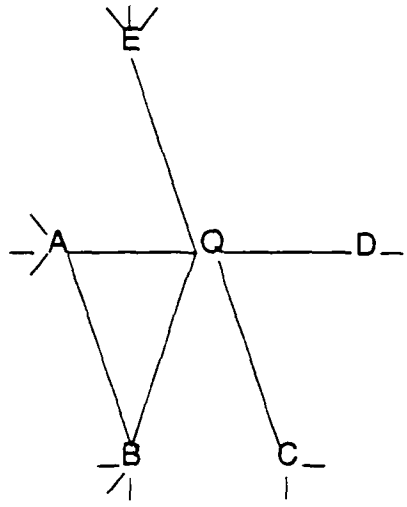

Figure 4.4: $L = 4$. Q, A, B active; C, D, E states irrelevant. Q has neighbors A, B that satisfy 2b

Lemma 4.7 *Assume that node Q in the network must invoke the Extended Selection Rule. The actions specified in the next paragraph are sufficient to cause some pair of nodes in the network to recognize each other. "Recognize each other" includes the obvious case (discussed in the next few paragraphs) where some pair of nodes explicitly recognize each other. It also will be taken to include the case where node Q explicitly determines that the node it recognizes can make a safe decision by invoking the primary selection rule.*

If Q must invoke the Extended Selection Rule, then Q is active and has no neighbor with one or more passive neighbors in common with Q. [Loosely speaking, all of Q's neighbors are also active.] Let Q choose (recognize) S if S has the maximum number of neighbors in common with Q. If there is a tie for S, let Q choose the S with the highest ID. Q must announce the node S in Q's next PROP, and must indicate that the node S was recognized, as opposed to excluded. Note: The maximum number of neighbors held in common between Q and S must be greater than one for this lemma to apply.

Proof: If Q is invoking the Extended Selection Rule, then the node S that it chooses (recognizes) is active. Active nodes must either invoke the Primary Selection Rule or the Extended Selection Rule. If S invokes the Primary Selection Rule, then S can immediately exclude some PR and will announce this fact. Q will hear it, and thus explicitly determine that "the node it recognizes can make a safe decision by invoking the primary selection rule". This satisfies one of the two cases of recognition.

If S invokes the Extended Selection Rule, then S also announces the ID of the node that it is trying to recognize. S will always be able to recognize at least one node, Q, because Q is a neighbor of S. If S announces Q, then this satisfies the obvious recognition case. If S announces some node other than Q, then we can inductively apply this lemma at the node that S announces. This induction continues until we reach two nodes that "recognize each other". Note that we create a chain of nodes in this case.

Each node recognizes a neighboring node, and that neighboring node recognizes one of its neighboring nodes.

Observe that it is impossible to create a circular chain with more than two nodes. The chain must have an end: no cycle can exist. If such a cycle existed, then either it consists of nodes that all have the same number of neighbors in common with the subsequent node, or it consists of nodes with a monotonically non-decreasing series of number of neighbors in common with subsequent nodes. The former is impossible because we chose to point to increasing ID numbers in case of a tie; the latter is impossible in a cycle.

We conclude that some two nodes will always point to each other. There may or may not be other directed arcs that point to those two nodes. This set of connected directed arcs is called a backbone. Graphically, a backbone is a tree with all arcs pointing toward the root. The root consists of the two nodes that recognize each other. It is possible for more than one backbone to exist, but they can not, and will not, intersect – no nodes will be held in common.

It is worth elaborating upon the consequences of this lemma. Whenever part 2a of the Extended Selection Rule must be invoked, lemmas 4.6 and 4.7 will be used. Therefore, at least one backbone must exist in the network. The length of a backbone – the maximum depth of its tree – is determined by the network configuration. If the length of the backbone is greater than one, then the decision that the first node(s) – the leaf or leaves – in the backbone must make is held up until the result of the decisions of successor node(s) are made available. The results propagate out from the core of the backbone. Why? Once the two core nodes (Q and S) of the backbone make their decisions, they become passive. Once Q (or symmetrically, S) is passive, all adjacent (active) nodes now have a passive node in common with some subset of the adjacent nodes. Now, Q's (and S's) neighbors may invoke the Primary Selection Rule because Q (and S) will be passive. In particular, any neighbors who may have been forced to wait because of Extended Selection Rule part 2b can now make a safe decision.

4.3 Stability and Convergence

Suppose that you freeze the network at time t. Run the heuristic to enable nodes to choose which links to keep. This is called a major cycle. Running the heuristic may require many exchanges of information. These exchanges are called minor cycles. At least one minor cycle is required so that all nodes can determine the state of their neighbor nodes at time t.

How long will it take for the heuristic to run? Since only active nodes require using the heuristic, it suffices to look only at those nodes. An active node can be classified as one of the following types:

0. All of its neighbors have exactly 0 neighbors in common with it.

   In this case, the node *must* locally partition the network. Thus an active node in this case can make its decision in one (minor) decision cycle.

1. None of its neighbors have more than 1 neighbor in common with it, and at least one (technically two) neighbors have at least 1 neighbor in common with it.

   If none of these neighbors are passive, the number of cycles depends on the network configuration. This is discussed below.

2. At least one of its neighbors has at least 2 neighbors in common with it.

   The heuristic will solve this case. However, if none of the neighbors are passive, then the number of cycles depends on the configuration. See below.

If there are backbones (see lemma 4.7 for definition), then it could take a number of (minor) decision cycles equal to the length of the longest backbone for the heuristic to converge to the final solution. Backbones may be present when active nodes that are classified as Type 2 are present.

If there are nodes that can be classified as Type 1, it is possible for the following situation to occur in some configurations. For brevity, we refer to the group of three nodes that fall into this category as dependent trios when all three nodes are active. [In Figure 4.5, nodes A, B, and C form a dependent trio.]

Consider a set (or sets) of three active nodes (A, B, C) with the following characteristics [see figure 4.5]:

- All of A's neighbor nodes (except for the other two nodes in the trio A-B-C) are completely partitioned from each other. There is no path of any length from one neighbor to another, other than through A. The same condition holds for B and C.
- In addition, nodes A, B, and C are connected by three links.
- These three links are the only way that nodes connected (directly or indirectly) to any one of the three nodes can communicate with nodes connected (directly or indirectly) to one of the other two nodes.

Since the nodes are active, they must drop one link. However, there is no way for the three of them to make a choice and keep all of the collective nodes connected.

Now, let us pick one of the three nodes (say A). Suppose that two of A's neighbors (other than B and C) are active. Call those two neighbors X and Y. Suppose further that a link existed between X and Y [See Figure 4.6].

Considering only the links between A, B, C, X, Y, it is impossible to have A, B, C, X, Y drop links and keep them all connected. Suppose, though, that X finds out that it can drop a link (other than the ones with Y or A) and the net will not become disconnected. Then X would do so and would revert to being passive. The links with Y and A remain. A and Y would be delighted to hear this news because they could then exclude each other (both would remove the A-Y link) [See figure 4.7].

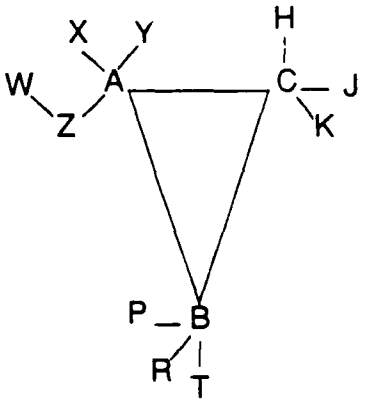

Figure 4.5: $L = 4$. A, B, C active.

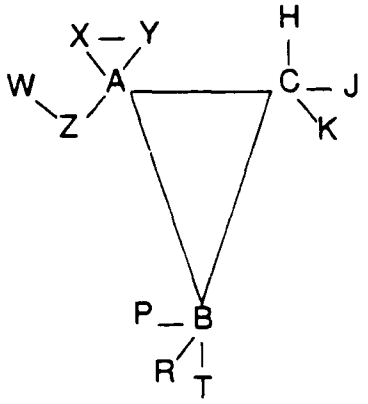

Figure 4.6: $L = 4$. A, B, C, X, Y active. Other links for X, Y not shown.

Each would revert to a passive state since each has made a decision. Next, B and C would be delighted to hear that A is passive. They could remove the B-C link [See Figure 4.8].

This is an example of a dependent chain. It consists of linked trios of nodes, where the nodes in each trio have each other as the most likely nodes to exclude. If any node in the chain turns passive (such as X did in the example), then the chain can be resolved safely.

Thus, if there are linked trios in the network, it could take a number of decision cycles equal to the length of the longest dependent chain for the heuristic to converge to the final solution.

If both linked trios and backbones are present, then convergence could take a number of decision cycles equal to the sum of the maximum number of cycles spent to resolve either separately. At worst, this number will be equal to one-third of the number of nodes in the network.

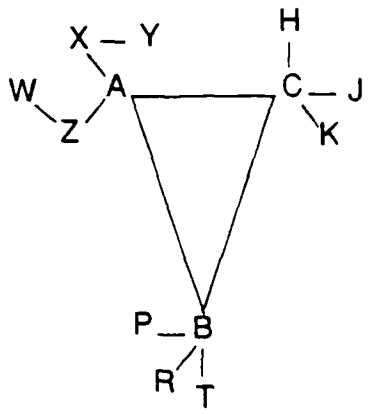

Figure 4.7: $L = 4$. B, C active. A, Y passive after removal of A-Y link. Other links for X, Y not shown.

Once this number of (minor) decision cycles has passed, we can allow the network to accept more links. We can then repeat the major cycle. The network will only be unconnectable based on local information if some active node is ultimately classified as Type 0, or Type 1 with no passive neighbors in common. Hence, each major cycle leads to a stable configuration.

Oscillations across major cycles are impossible because a node only considers a non-good neighbor node as a candidate if that node does not appear in the current routing tables. For a candidate to be accepted, a node only removes one of the existing L links if it can determine that the link can be removed without losing a path to the node on the other side. If this can not be determined, the candidate is not accepted. Thus, each node will only remove links if it can strictly increase the number of nodes it (and the network, or the current partition) can reach.

4.4 Speeding Up Convergence

The term backbone was defined in Lemma 4.7, Section 4.2. It was shown that links could be removed in a way that always kept the network connected. However, if Lemma 4.7 is implemented in the manner in which it was presented, then the speed at which the network stabilizes becomes an issue. Depending on the length of the backbone, it could take many cycles for the chain reaction to reach an end.

The speed of convergence also depends upon the length of dependent trio chains, as was shown in the previous section. Resolving such a chain also requires a chain reaction, so to speak.

It would be desirable if all of the nodes along the backbone could be forced to act simultaneously, or nearly so. The 2-cycle resolution process accomplishes this goal in two cycles, regardless of the length of the backbone chain(s).

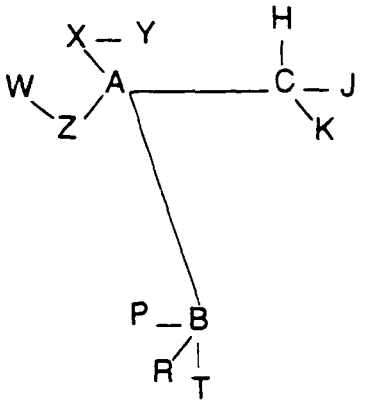

Figure 4.8: $L = 4$. B, C now passive after removal of B-C link. Other links for X, Y not shown.

Resolving dependent trio chains quickly is also desirable. Unfortunately, it is not possible. The inability to quickly resolve this situation is not a critical problem, though. In any real network configuration, a large L implies that long dependent trio chains are unlikely to occur.

2-Cycle Resolution Process:

The 2-cycle resolution process is invoked only by active nodes that have no neighbor with one or more passive neighbors in common, but have at least one neighbor with at least two neighbors in common. Q and S (see Lemma 4.7, and also Figure 4.3) will run this process, as will any other active node which satisfies the prerequisite conditions.

Note: This process is used only when indicated by the **Extended Selection Rule**.

For simplicity, this algorithm is run from "your" point of view. One may substitute "Q [is]" for the word "you [are]", and "Q's" for the word "your". Your (Q's) recognized node ($R.N$) is your neighbor with the largest number of neighbors in common with you. In case of a tie, pick the highest ID from the set of nodes that tied. Note: In the nomenclature of Lemma 4.7, Q's $R.N$ is S.

There are only two cases for you to choose between.

1. If your $R.N$ explicitly recognizes you, then you have mutually recognized each other. You are the core of a backbone.
   Action: Apply lemma 4.6.
   Note: All of the nodes in the set $N.IC$ are *bystanders* to your decision. Bystanders can determine that you and your $R.N$ have mutually recognized each other.
2. The (active) $R.N$ that you recognize doesn't explicitly recognize you.
   Action: Arbitrarily exclude some node in the $N.IC$ set that corresponds to the link between you and your $R.N$. The only restriction is that you cannot exclude a node that has you designated as its $R.N$.

It should be apparent why this is called the two-cycle resolution process. All nodes have already exchanged information in PROPs. At this point, all nodes know whether their neighbors are active or passive. In the first cycle, each active node indicates that it is excluding some node via the Primary Exclusion Rule, or it announces its RN. After this exchange, all active nodes running this process figure out whether case I or II applies and perform the associated action. The decisions are broadcast in the second cycle. After this exchange, some bystanders have been excluded (and become passive), and the others can unilaterally exclude a node since passive neighbors are present. Of course, it takes a third cycle for everyone to find out that every node is passive, but no decisions are still outstanding.

It may not be apparent why this process works. Below, the two possible states are reproduced. It is shown that the net remains connected.

Correctness Proof of 2-Cycle Resolution Process:

1. If your RN explicitly recognizes you, then you have mutually recognized each other. You are the core of a backbone.
   (a) You are not also a bystander; no two of your immediate neighbors are each other's RNs.
   It is obvious that Lemma 4.6 will keep you, your RN, and the NIC set defined by your choice of RN [i.e., $NIC(you, yourRN)$] connected. For your other active neighbor nodes:
      i. If you are designated as its RN, that node will not break the link between you and it, since it is part of the backbone.
      ii. If you are not its RN, then it cannot break the link with you because:
         A. you have no passive neighbors in common with it, so it cannot use the Primary Selection Rule to exclude you, and
         B. its RN, if it has one, is not one of your neighbors, so you are not one of its bystanders.
   (b) You are also a bystander.
   In this case, your recognized node has recognized you, and you have links with one or more pairs of nodes that recognize each other (See Figure 4.9[4]). Any decision made by a pair of nodes that recognize each other will cause you to relinquish at most one of those two links. Your own decision will only cause you to delete one link. Even if your decision causes you to delete a link that some pair thought that you would keep (via their mutual negotiation from lemma 4.6), yet another path among that pair of nodes, you, and your recognized node must exist. (In Figure 4.9, both the SB link and the QB link will be deleted. However, Q, S, and E can still reach B and A via C.) This is true because not only were you a bystander to their decision, but one of the pair of nodes had to have been a bystander to your decision as well. The net remains connected.

[4]About Figure 4.9: Technically, QS and BC cannot recognize each other by the rules given in this paper unless $L = 5$ and some additional nodes are added. However, defining $L = 4$ and allowing QS and BC as pairs does not make a difference and results in a less cluttered diagram.

2. The (active) $R.N$ that you recognize does not explicitly recognize you.

   (a) The $R.N$ you designated has an excludee by the Primary Selection Rule.
   The $R.N$ you designated has some neighbor with one or more passive neighbors in common. That neighbor is not you, nor any other neighbor in your NIC set. Therefore, the link between you and your $R.N$ will remain.
   The neighbor that you choose to exclude will either be waiting to be excluded, or will make a safe exclusion. It will not exclude your $R.N$ unless it can guarantee that it has an alternate path that does not go through you.

   (b) The $R.N$ you designated has an $R.N$.
   The $R.N$ that you designated has no neighbors with passive neighbors in common with itself. The $R.N$ that you designated has at least one neighbor (you) with two neighbors in common, because you were able to designate it as an $R.N$. Your $R.N$ will also be running this procedure. The net remains connected by virtue of the Backbone Lemma that follows.

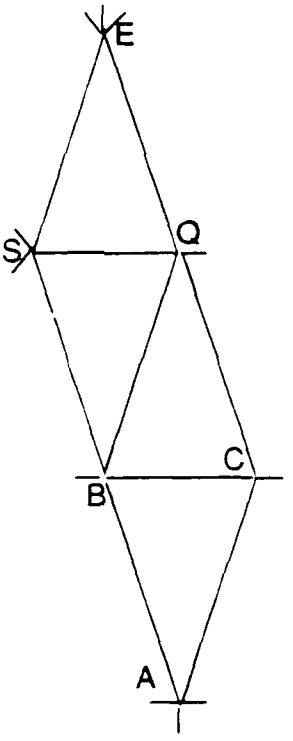

Figure 4.9: $L = 4$. E, S, Q, B, C, A active: Q, S mutually recognized. Q agrees to keep B, S agrees to keep E. B, C mutually recognized. B agrees to keep A, C agrees to keep Q.

Backbone Lemma:

Let S be Q's $R\mathcal{N}$, and T be S's $R\mathcal{N}$. We have already shown that there can be no cycles of length greater than two in a backbone. If Q and T are the same node, then Q and S mutually recognized each other; lemma lemma 4.6 and 4.7 solve that problem. We confine ourselves to the case where Q and T are distinct nodes.

Q can always break a link with one of the neighbors in its $\mathcal{NIC}$ set, as long as Q doesn't break a link with a node that considers Q as part of the backbone.

Proof: From Q's point of view, the set $\mathcal{NIC}(S,T)$ either has

1. no nodes in common with the set $\mathcal{NIC}(Q,S)$, or
2. at least 1 node in common with the set $\mathcal{NIC}(Q,S)$.

Q can not always tell which of these two alternatives is actually true; it does not matter. In case 1, S will always break a link with a node outside of $\mathcal{NIC}(Q,S)$. Regardless of whether S and T cooperate, S breaks a link with a node in the set $\mathcal{NIC}(S,T)$. It is clear that Q can then break a link with any node in $\mathcal{NIC}(Q,S)$ without disconnecting the network.[5]

In case 2, consider a node J held in common with both $\mathcal{NIC}$ sets. It must be the case that J has 3 connections to this section of the backbone – one to Q, S,and T. If Q excludes J, 2 links to the backbone remain. If S and T cooperate, then J will remain connected to at least one of them. If S and T aren't cooperating, then S could also exclude J. T cannot exclude J, unless J is also held in common with T's $\mathcal{NIC}$ set – $\mathcal{NIC}(T,*)$ where * is not Q, S, or T. Induction shows that J will never be severed from the backbone, since the final two backbone nodes always cooperate.

It can be seen that the two-cycle resolution process allows us to speed up the convergence enormously. In fact, convergence to a stable solution can be guaranteed in only two cycles if we ignore any dependent trio chains that might exist. In passing, it is noted that some A, B, C trios can be resolved properly in one cycle, because one of the trio members (say B) may have announced an excludee node or a $R\mathcal{N}$ node. The other two, A and C, would hear this simultaneously, and could exclude each other. A, B, and C all remain locally connected. A prudent approach for trios with no one-cycle resolutions would be to simply exclude the candidate.

4.5 Implementation in SURAN

Some procedures and terms have not been accurately defined. Many of these could be considered implementation details and, thus, have been relegated to this section. However, two restrictions make the heuristics in this paper appear to be rather difficult to implement in the current SURAP 1.0 protocols. The first restriction is that we assumed that all communication was reliable. The second restriction is that all nodes in the network be synchronized. Before explaining terms used earlier in Section 4 and specifying some implementation details, it is necessary to dispose of these two restrictions.

[5] Implementation Note: It might be especially desirable for Q to break with a node in $\mathcal{NIC}(Q,S)$ which reported a max "$\mathcal{NIC}$" of one, since that node is waiting for a decision (from some node) anyway and cannot be a part of the backbone.

Unreliable communication is rather easy to deal with, at least in a probabilistic sense. Do not send information out in only one PROP. Send the information in several PROPs. Increasing the number of PROPs increases the likelihood that the PROP is received by all intended recipients. All PRs would have identical repeat counts. This approach is currently taken when it is necessary to notify neighbors about bad routes.

Synchronization is trivial to deal with. It is not necessary. It was an artifice introduced to talk about convergence and stability, and to simplify the initial presentation of the heuristic. Consider that passive nodes do not run either the Primary or Extended Selection rules. They do not need to be synchronized. An active node that can use the Primary Selection rule does not need to be synchronized with any other node since there is at least one passive neighbor that the active node depends upon. Finally, an active node that must use the Extended Selection rule will only do so because all of the relevant neighbors are active. If they are all active, then the relevant nodes will also be running either the Primary or Extended Selection rules.

4.5.1 Expungability

We define an expungable neighbor as one that appears in a PR's neighbor table and that is neither a good neighbor, nor a neighbor with which a link is in the process of being brought up for the first time. This differs slightly from Miller's bad neighbor definition in rule 0. We consider the neighbor associated with a link for which no quality value has been reported and which was not brought up in a reasonable amount of time as a bad neighbor, and therefore expungable; his rule doesn't.

The above definition implies that we should be able to tell when a link is in the process of being brough, up for the first time. In addition, it is advantageous to indicate when too much time has been spent trying to bring up a link without success. Thus a link in the process of being brought up for the first time is not expungable, unless too much time has gone by.

Links that are being brought up for the first time can be distinguished by counting PROPs. When a new neighbor is added to the neighbor table, initialize an associated counter to some predetermined value. This value should be somewhat greater than the minimum number of PROPs that must be exchanged before the quality builds up to an acceptable level. Each time that a PR broadcasts a PROP, it should decrease the counter for every neighbor (but not below zero). If the counter has reached zero and the neighbor has not been marked good, then the neighbor is expungable. It is important to update the counter on broadcast PROPs instead of received PROPs so that missed PROPs do not adversely affect the counters. We prefer a counter approach to a timer approach because timers are burdensome to maintain. The neighbor table will need expanding to keep track of these quantities for each link.

The reason to distinguish expungable neighbors from non-expungable neighbors is that we intend never to delete entries from the neighbor table unless necessary. We only consider it necessary to delete an entry if the neighbor table is full and we hear a PROP from a PR not currently in the neighbor table.

4.5.2 Active/Passive States

The definitions for passive and active neighbors remain the same. However, these distinctions are only useful if a node's neighbors know the current state of that node. If these suggestions are to be implemented, we propose that nodes announce their state in each PROP. If the maximum neighbor size is identical for all PRs, an active PR can be distinguished by counting the number of immediate neighbors reported in its PROP. However, we prefer an explicit indicator so that radios with differing amounts of memory can have different maximum neighbor table sizes.

A node announces that it is active when it obtains a valid candidate, and remains active until that link becomes good, or until that link has spent too much time being brought up. In the former case, it attempts to exclude a node based upon the exclusion rule(s); in the latter case, it simply drops the candidate. In either case, it becomes passive when it drops a node.

We propose that when a node changes from the active to the passive state it should remain passive for several PROP cycles, even if it hears a PROP from a PR that should become a candidate. This will increase the likelihood that its neighbors will find out the correct state. Of course, during that time, it should not fill the candidate slot. The neighbor table should be expanded to keep track of the state of each neighbor (passive or active). Provision must be made to keep track of a PR's own state, and the number of PROP cycles during which the node must remain passive before being allowed to change to active.

One should also keep track of the number of passive neighbors in common in the neighbor table on a per-node basis. There are two ways to do this. The number of passive neighbors held in common can be updated for a particular neighbor either only upon receipt of that neighbor's PROP or potentially upon the receipt of every PROP. Because PRs do not keep track of the neighbors of our immediate neighbors and PROPs can be missed, a tradeoff is involved in choosing one method or the other. Neither updating methods will be 100% accurate all of the time. The first method might reflect reality only for a single instant in each PROP cycle. The second method attempts to reflect reality continuously, but it is possible for it to be wrong upon occasion.

The first method causes updates for a PR to occur only on the receipt of that PR's PROP. Updating only upon receipt of that PR's PROP implies that changes in the state of common neighbor nodes are only perceived once each PROP cycle. This means that the information in the table isn't always as current as possible.

The second method causes updates to occur upon the receipt of every PROP. In this case, the number of passive common neighbors can be kept current by adjusting all affected counters by plus or minus one whenever a received PROP indicates that its sender changed from active to passive, or from passive to active, respectively. The affected counters are the counters for all neighbors in a PR's neighbor table that appear at level one in the received PROP. As in the first method, the PR explicitly sets the counter for the sender of the PROP. This is necessary to correct errors that may have been erroneously introduced by using PROP data from a new neighbor in common. The following example shows why this is necessary by presenting two cases that are indistinguishable from the point of view of PR A.

Example

Assume $t_1 < t_2 < t_3$

At instant t_1, PR A knows that PR B has 2 passive neighbors in common (say C or D; A doesn't keep track of which neighbors they are). At instant t_2 a PROP from C is heard and C indicates that it is active. A realizes that C was passive before, and that B is listed in C's PROP and decrements the counter associated with B from 2 to 1. This is proper and correct.

Suppose that at t_1, PR X was a good neighbor of A, but not of B. Further suppose that sometime between instant t_1 and t_3, X and B become good neighbors. At t_3, A hears a PROP from X. If X were passive at t_1 but active at t_3, then A will err in changing the counter from 1 to 0. This error occurs because A cannot distinguish this situation from the one in which A received a PROP from C. In reality, the counter should not be changed, since the true number of passive neighbors held in common is still 1 (neighbor D).

4.5.3 Candidate Indicator

When a node has a candidate neighbor, that neighbor must appear in the PROP so that link quality information can be exchanged. It would be preferable if the PROP entry for a candidate was specially marked so that other nodes will not use that node in any alternate path calculations. Other nodes would not consider that node because it normally does not have good link qualities in both directions. However, it is better to explicitly indicate the candidate in order to minimize future problems, and to keep a rare boundary condition from causing any temporary problems.

The candidate slot could be a known entry in the neighbor table, or the slot could be separately maintained. As previously stated, the candidate PR must appear in the PROP, regardless of where it is stored.

5. Areas for Further Research

In the **Extended Selection Rule** of Section 4.2, we stated that the candidate should be excluded for routing stability. However, this guarantees that the candidate is locked out from our partition unless changes occur elsewhere. If no further changes occur, the candidate and all nodes in its partition are permanently locked out from the network. Some network topologies, like the repeater-on-a-hill topology, do not allow for a non-partitioned network given neighbor table size constraints. Sometimes, though, the heuristic may have simply chosen the wrong set of neighbors. Continuous perturbation of the non-optimal solution by accepting the candidate and randomly rejecting some other neighbor will eventually lead to some connected solution if the topology allows it. This perturbation, though, causes permanent instability for repeater-on-a-hill topologies.

It is possible that both of these problems can be avoided by allowing unconfirmed routes. Currently, a reporting PR is given for each PRID that can be reached via a good route. A good route is one which possesses logical links at every hop. However, it is possible to reach a PR as long as physical links exist at every hop. We suggest that it is possible to implement a last-ditch mechanism that will coexist with whichever logical neighborhood heuristic is implemented. The idea behind this mechanism is to indicate that an unconfirmed route may exist to PRs that can not be reached via a good route.

We believe that only minor changes to the routing table and in PROP content are necessary. Currently, the routing (tier) table entry for a PR contains a reporting PR, its tier level, and a good/bad route indicator. Adding an 'unconfirmed' indicator may allow the PRs to route to neighboring PRs even when there is no room in the neighbor table for a logical link. The PROPs would also need to be expanded to distinguish between IDs of PRs with good routes and IDs of PRs with routes that are unconfirmed. This requires very little additional code, and does not require much expansion of the routing table or the PROP packet.

Under this scheme, we would continue to reject the candidate for routing stability if no safe exclusion were possible. However, all of the (non-reachable) PRIDs in the PROP would be added to the routing tables and flagged as unconfirmed routes. The candidate would be listed as their reporting PR. Packets destined for a PRID with an unconfirmed indicator are sent to its reporting PR. Failure would cause the route to be marked as bad; success does not change the indicator.

The tier routing update algorithm should operate under the presumption that a good route of any length is preferred to any unconfirmed route. Note that a good indicator should only be changed to bad, although a bad or unconfirmed indicator can be changed to either of the other two indicators as circumstances demand. For example, an unconfirmed indicator should be treated just like a good indicator when a PROP is received with PRIDs in the bad tier data: those PRIDs for which the PROP receiver has the

PROP sender listed as the reporting PR should have the indicator changed to bad. The changes that ought to occur in all other situations follow from the presumption of good over unconfirmed.

More thought needs to be given to this approach, but it appears to have a good deal of potential for maintaining connectedness when using tier routing.

Acknowledgements

I am indebted to Dr. Greg Lauer for the many discussions we held and for critiquing several versions of this paper. His ability to focus in on weak areas in the early formulations was invaluable. I would also like to thank Marian Nodine and Jim Ong for their review efforts. Both of them spent hours working through the examples and lemmas, and made many useful suggestions.

Bibliography

[1] Y. Dalal. A distributed algorithm for constructing minimal spanning trees in computer-communication networks. In *Proceedings of the Fifth Texas Conference on Computing Systems*, October 1976.

[2] F. Glover and D. Kingman. Finding minimum spanning trees with a fixed number of links at a node. In B. Roy, editor, *Combinatorial Programming: Methods and Applications*, September 1974.

[3] B. Miller. *Limiting Logical PR Neighborhood Size.* Technical Report SRNTN-43, Rockwell, Inc., 1986.

[4] J. Tornow. *Functional Summary of the DARPA SURAP1 Network.* Technical Report SRNDOC-9, SRI, September 1986.